下载APP

15 | 流量隔离：MySQL数据库隔离是怎么做的？

2021-11-20 高楼

《全链路压测实战30讲》 课程介绍 >

讲述：高楼
时长 26:23 大小 24.16M

你好，我是高楼。

这节课，我们详细讲讲怎样基于微服务技术进行 MySQL 数据库隔离的落地。

对线上压测来说，最重要的环节就是**流量隔离**了，而这其中最核心的又是**数据库隔离**，只有识别并隔离正式流量与压测流量才能避免产生**脏数据**。

为什么要做数据库隔离？

为什么做数据库隔离呢？我给你举个例子。记得几年前，我当时所在的公司做线上压测。当时团队并没有做数据库隔离，而是直接在线上生产库进行压测，导致产生了很多脏数据。结果花费了很长时间去清理数据，但最终也没有完全清理干净，它不仅污染了生产数

据还对线上业务产生了不好的影响，影响了用户体验。**所以说做线上压测，数据库隔离是必须做的一个环节。**

当然了，如果你的公司不差钱，也可以直接隔离出一套线下 1:1 镜像环境做压测。但是大部分公司并没有这么“土豪”，还得使用目前的生产环境做压测，那么数据库隔离技术就必不可少了。

首先，我梳理了一下目前业界对于数据库隔离的主要解决方案，以及它们的优缺点和适用场景。

	数据偏移	影子库	影子表
优点	• 实现简单，无须改造中间件	• 物理隔离 • 对业务数据无侵入性 • 安全性高，对生产库无任何影响 • 扩展性较强 • 数据清理简单	• 逻辑隔离 • 中间件容易支持 • 安全性一般，对生产库有一定影响 • 扩展性较强 • 数据清理简单
缺点	• 数据进入生产表 • 对业务数据侵入性较大，会影响表结构设计 • 数据清理复杂，需要根据每个字段的标记单独制定清理规则 • 扩展性较差，新的字段需要设计新的标记 • 安全性较差，设计逻辑有纰漏，会影响生产数据 • 压力大的情况下，会挤占数据库的资源	• 数据库资源双倍 • 需要切换数据库连接，中间件改造有成本	• 压力大的情况下，会挤占数据库的资源
适用场景	• 流量标记没打通 • 技术体系不完整，改造成本和复杂度过高	• 全服务流量标记 • 资源充裕 • 中间件技术体系完整	• 全服务流量标记 • 中间件技术体系完整

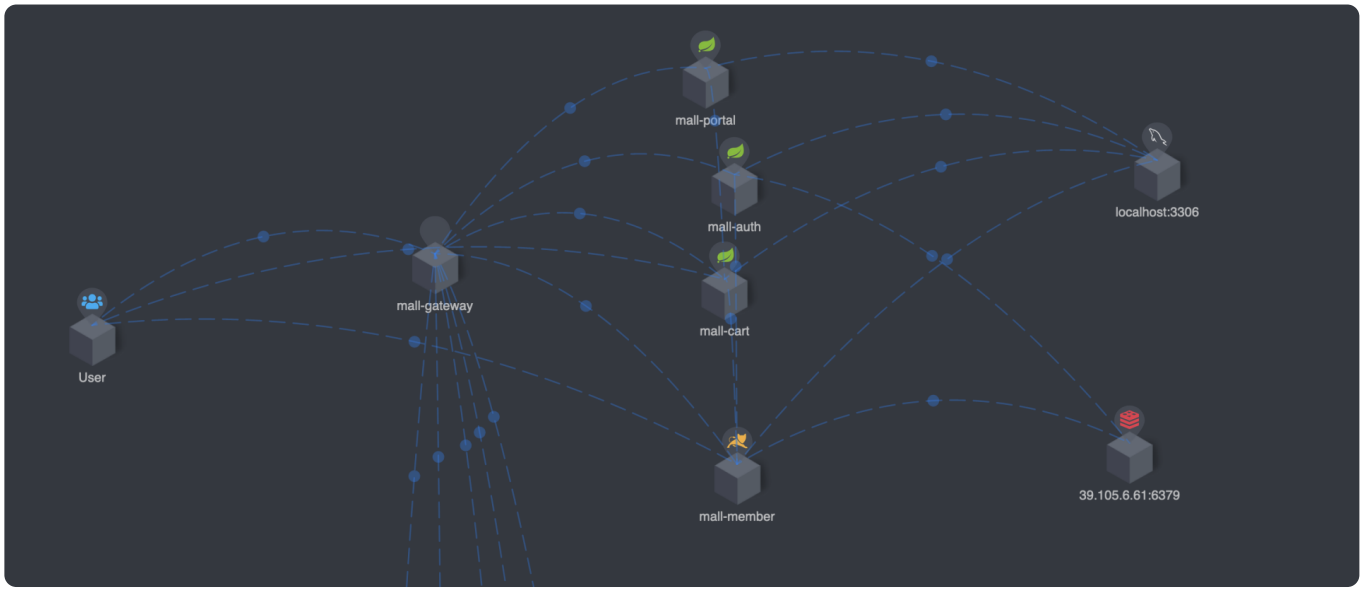
极客时间

我们可以看到，根据不同的项目情况，可以选择不同的技术方案，**这里最优、最安全的方案当然首推影子库**，具体的优缺点上面表格已经写得非常清楚了。

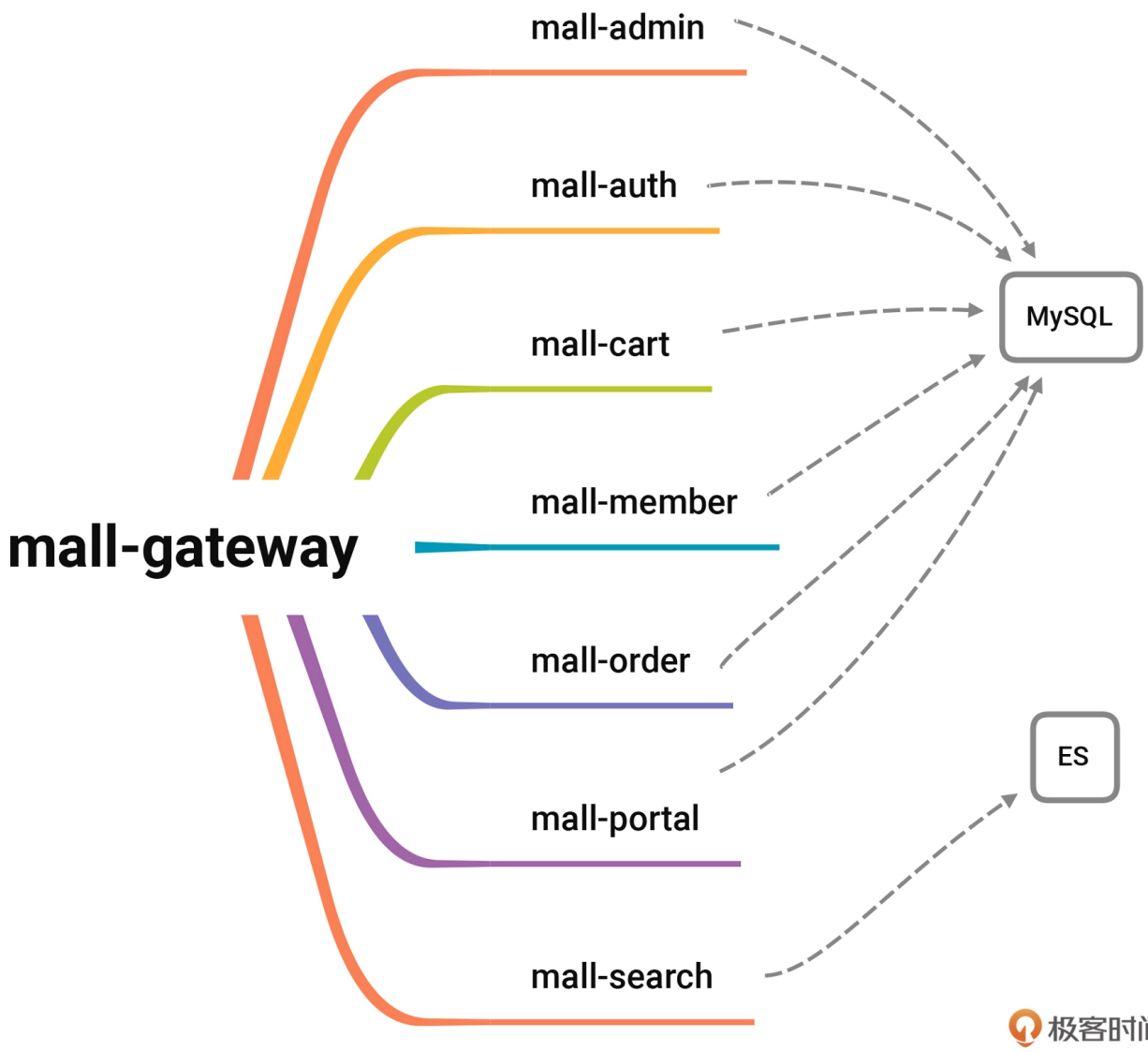
其实，上面这三种数据库隔离技术现在都已经很成熟了，要做到并不太难。但是出于各方面的原因，我们在网上很少看到过完整的技术解决方案，而有些公司把它们视作“独门秘方”，所以数据库隔离还远远谈不上普及。不过随着市场的发展，数据库隔离的面纱迟早会缓缓揭开。

对于我们这个电商项目，我选择采用影子库。因为数据偏移缺点很明显，不值得选择，而影子表代码修改量大，风险大不好控制。影子库呢，风险小，改动量小。综合权衡后，我们认为采用影子库性价比更高。

在进行影子库的改造之前，让我们先来看下目前项目的系统链路图。心中有链路图，才能知道在哪些地方改造。



上面是简单的链路图，你可以很清楚地知道，哪些服务与 MySQL 数据库有关系。为了方便你更直观地理解，我给你画了个思维导图。



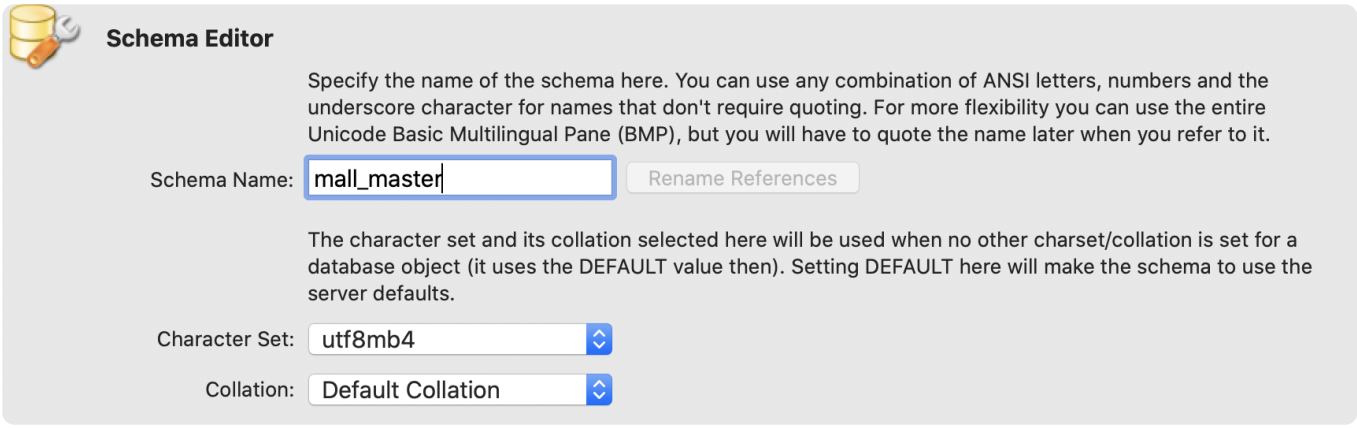
有了链路图做指导，接下来就是修改各个模块，进行具体的代码改造工作了。为了减少风险，我们可以先写一个 demo 做分库实例演示。如果 demo 没问题，再修改一个线上实例做实验。验证完毕并权衡风险后，再同步其他服务，只有这样反复验证才能把改造风险降到最低。

影子库技术预演

在做影子库改造之前，需要调研可以用什么技术做，心中有思路才好下手实践。

准备工作

首先，打开开发工具（如 Idea）新建 SpringBoot 工程（说明：这里用什么工具与什么工程无所谓，只要达到分库的目的就行）。既然是分库工作，就需要在 MySQL 数据库中新建两个数据库，如下图：

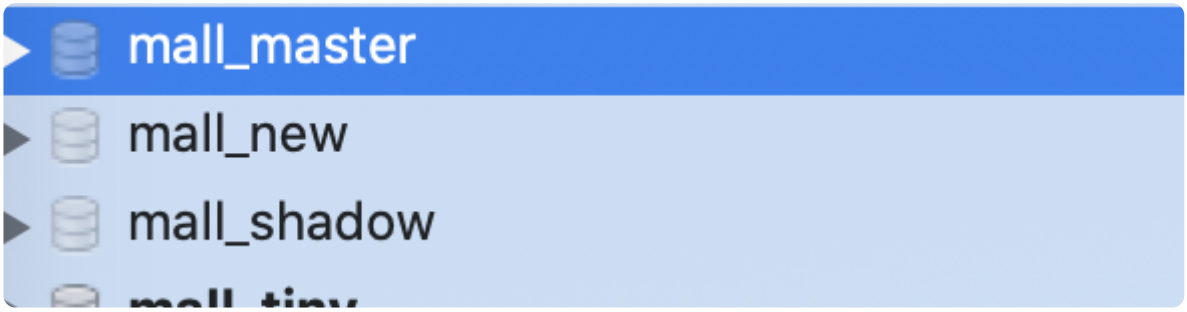

Schema Editor
Specify the name of the schema here. You can use any combination of ANSI letters, numbers and the underscore character for names that don't require quoting. For more flexibility you can use the entire Unicode Basic Multilingual Pane (BMP), but you will have to quote the name later when you refer to it.
Schema Name:
mall_master
Rename References
The character set and its collation selected here will be used when no other charset/collation is set for a database object (it uses the DEFAULT value then). Setting DEFAULT here will make the schema to use the server defaults.
Character Set:
utf8mb4
Collation:
Default Collation

新建数据库参考语句如下：

复制代码

```
--正式库
CREATE SCHEMA `mall_master` DEFAULT CHARACTER SET utf8mb4 ;
--影子库
CREATE SCHEMA `mall_shadow` DEFAULT CHARACTER SET utf8mb4 ;
```

新建成功后如下图：



到这里，数据库就建完了，但是光有库没有表是不行的，所以需要在两个新的数据库中新建一样的表，参考语句如下：

复制代码

```
DROP TABLE IF EXISTS `ums_admin`;

CREATE TABLE `ums_admin` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `user_name` varchar(64) DEFAULT NULL COMMENT '用户名',
  `pass_word` varchar(64) DEFAULT NULL  COMMENT '密码',
  `icon` varchar(500) DEFAULT NULL COMMENT ' Header 像',
  `email` varchar(100) DEFAULT NULL COMMENT '邮箱',
  `nick_name` varchar(200) DEFAULT NULL COMMENT '昵称',
  `note` varchar(500) DEFAULT NULL COMMENT '备注信息',
```

```
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  `login_time` datetime DEFAULT NULL COMMENT '最后登录时间',
  `status` int(1) DEFAULT '1' COMMENT '帐号启用状态：0->禁用；1->启用',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='管理员用户表';
```

打开项目，在 resources 目录下新建 application.yml 文件，因为这次 demo 的目的是要使用两个数据源做数据库隔离，所以要配置两个数据源。你可以参考我给你提供的 application.yml 配置文件。

复制代码

```
spring:
  datasource:
    master: # 数据源1
      url: jdbc:mysql://localhost:3306/mall_master?characterEncoding=utf8&useU
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver
      type: com.alibaba.druid.pool.DruidDataSource
      initialSize: 5
      minIdle: 5
      maxActive: 20
      maxWait: 60000
      timeBetweenEvictionRunsMillis: 60000
      minEvictableIdleTimeMillis: 300000
      validationQuery: SELECT 1 FROM DUAL
      testWhileIdle: true
      testOnBorrow: false
      testOnReturn: false
      poolPreparedStatements: true
      #    配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙
      filters: stat,wall,log4j
      maxPoolPreparedStatementPerConnectionSize: 20
      useGlobalDataSourceStat: true
      connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=
    shadow: # 数据源2
      url: jdbc:mysql://localhost:3306/mall_shadow?characterEncoding=utf8&useU
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver
      type: com.alibaba.druid.pool.DruidDataSource
      initialSize: 5
      minIdle: 5
      maxActive: 20
      maxWait: 60000
      timeBetweenEvictionRunsMillis: 60000
      minEvictableIdleTimeMillis: 300000
      validationQuery: SELECT 1 FROM DUAL
```

```
38        testWhileIdle: true
39        testOnBorrow: false
40        testOnReturn: false
41        poolPreparedStatements: true
42        #    配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙
43        filters: stat,wall,log4j
44        maxPoolPreparedStatementPerConnectionSize: 20
45        useGlobalDataSourceStat: true
46        connectionProperties: druid.stat.mergeSql=true:druid.stat.slowSqlMillis=
```

有了数据源配置文件，还需要读取配置文件的配置类。如果是单库实例数据源，用下面的代码读取数据源配置，项目就可以直接操作数据库了。

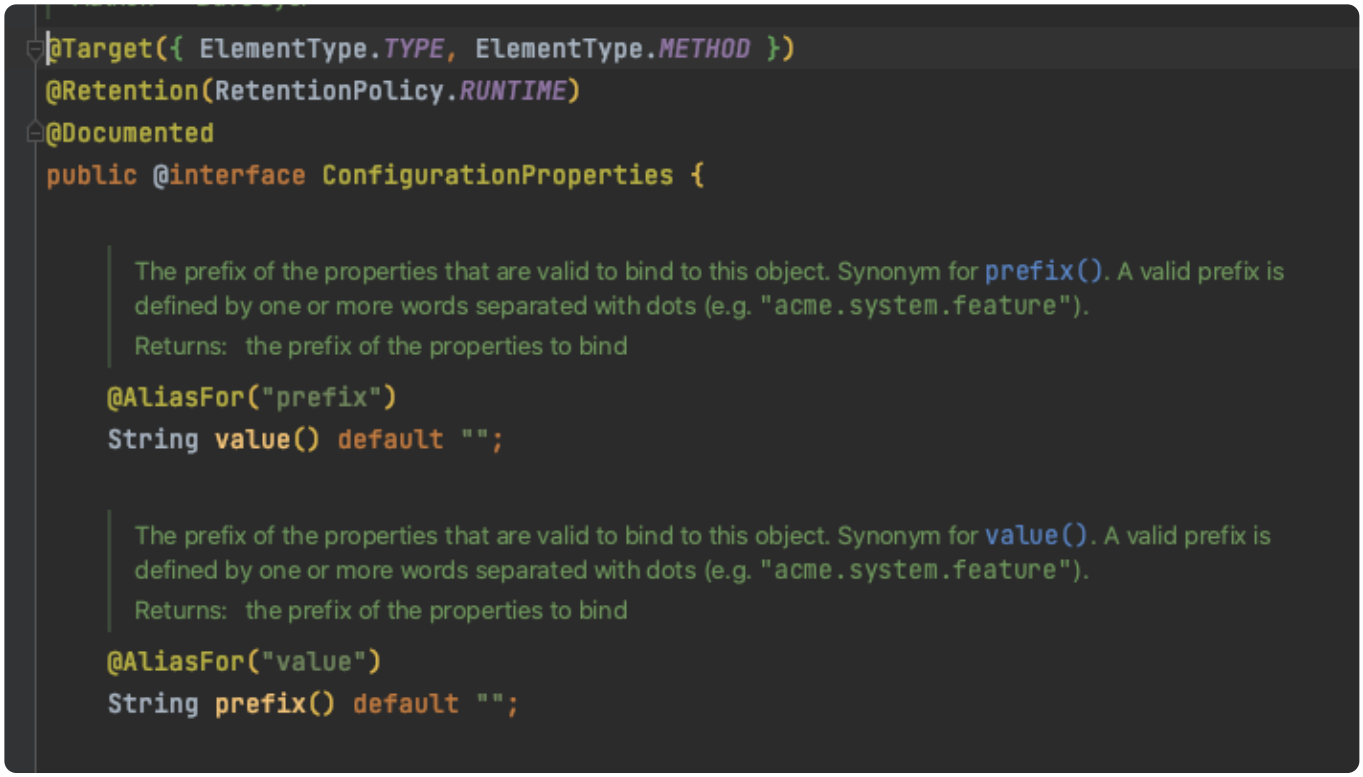
复制代码

```
1     @ConfigurationProperties(prefix = "spring.datasource")
2     @Bean
3     public DataSource druid() {
4         return new DruidDataSource();
5     }
```

但是如果是多库数据源，就不能按这个方式读取数据源了。那么应该怎么读取数据源呢？我们来仔细看一下下面的代码。

SpringBoot 在读取配置文件时，大量采用 @ConfigurationProperties 注解，从代码中可以得知，单个读取实例采用这一注解就可以读取到数据配置文件。

我们还可以通过查看源码知道，该注解带了一个 “prefix ” 属性，可以使用该属性读取数据源。另外，为了区分数据源，可以给不同读取数据源取不同的方法名字。


@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface ConfigurationProperties {
The prefix of the properties that are valid to bind to this object. Synonym for prefix(). A valid prefix is defined by one or more words separated with dots (e.g. "acme.system.feature").
Returns: the prefix of the properties to bind
@AliasFor("prefix")
String value() default "";
The prefix of the properties that are valid to bind to this object. Synonym for value(). A valid prefix is defined by one or more words separated with dots (e.g. "acme.system.feature").
Returns: the prefix of the properties to bind
@AliasFor("value")
String prefix() default "";

你可以参考我的代码。

复制代码

```
1  package com.dunshan.data.config;
2
3  import com.alibaba.druid.pool.DruidDataSource;
4  import com.alibaba.druid.support.http.StatViewServlet;
5  import com.alibaba.druid.support.http.WebStatFilter;
6  import lombok.extern.log4j.Log4j2;
7  import org.springframework.boot.context.properties.ConfigurationProperties;
8  import org.springframework.boot.web.servlet.FilterRegistrationBean;
9  import org.springframework.boot.web.servlet.ServletRegistrationBean;
10 import org.springframework.context.annotation.Bean;
11 import org.springframework.context.annotation.Configuration;
12 import org.springframework.context.annotation.Primary;
13 import javax.sql.DataSource;
14 import java.util.HashMap;
15 import java.util.Map;
16
17 /**
18  * @author dunshan
19  * @description: 数据库配置文件
20  * @date 2021-08-15 11:01:31
21  */
22
23 @Log4j2
24 @Configuration
25 public class DynamicDataSourceConfig {
26
27     /**
```

```
28      * 创建 shadow 数据源
29      */
30     @Bean(name = "shadowDataSource")
31     @ConfigurationProperties(prefix = "spring.datasource.shadow")
32     public DataSource shadowDataSource() {
33         return new DruidDataSource();
34     }
35
36     /**
37      * 创建 master 数据源
38      */
39     @Bean(name = "masterDataSource")
40     @ConfigurationProperties(prefix = "spring.datasource.master")
41     public DataSource masterDataSource() {
42         return new DruidDataSource();
43     }
44
45 }
```

数据源配置读取成功后，还得去思考怎么把数据源注入到上下文中，只有成功注入到上下文中才能使用该数据源，操作数据库中的表。但是注入成功后，要根据什么切换数据源呢？

在切换数据源之前，我们先来回顾一下之前做的 demo。我们做 demo 是为了识别流量标记，那怎么产生流量标记呢？

我们只需要在 Http 请求中增加 Header 信息，业务层会根据 Header 信息判断特定标记来切换数据源，如 JMeter 中 HTTP Header Manager 就可以增加 Header 信息标记：

数据源上下文的实现

在目前的 Spring 开发中，多多少少都会涉及 AOP（Aspect-Oriented Programming：面向切面编程)，这里只用其中一个特性，大家慢慢往下看就知道了。

另外，在切换数据源中还需要使用一个技术就是 ThreadLocal。但是因为 ThreadLocal 有点缺陷，所以我们这个场景使用 TransmittableThreadLocal 做数据源保存对象信息。这个在 第 13 讲已经讲得很清楚了，我就不多赘述了。

动态数据源 TransmittableThreadLocal 的代码实现，你可以看看我给出的例子。

复制代码

```
package com.dunshan.data.config;

import com.alibaba.ttl.TransmittableThreadLocal;
import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;
import javax.sql.DataSource;
import java.util.Map;

/**
 * @author dunshan
 * @description: 动态数据源切换
 * @date 2021-08-15 12:35:14
 */
public class DynamicDataSource extends AbstractRoutingDataSource {

     private static final TransmittableThreadLocal<String> contextHolder = new

    /**
     * 配置DataSource, defaultTargetDataSource为主数据库
     */
    public DynamicDataSource(DataSource defaultTargetDataSource, Map<Object, O
        super.setDefaultTargetDataSource(defaultTargetDataSource);
        super.setTargetDataSources(targetDataSources);
        super.afterPropertiesSet();
    }

    @Override
    protected Object determineCurrentLookupKey() {
        return getDataSource();
    }

    public static void setDataSource(String dataSource) {
        contextHolder.set(dataSource);
    }

    public static String getDataSource() {
        return contextHolder.get();
    }

    public static void clearDataSource() {
        contextHolder.remove();
    }
```

```
42 }
43
```

接下来要把数据源注入上下文中，只有这样数据源才能在系统中灵活使用。

复制代码

```
1     /**
2      * 如果还有数据源,在这继续添加 DataSource Bean
3      */
4
5     @Bean
6     @Primary
7     public DynamicDataSource dataSource(DataSource masterDataSource, DataSourc
8         Map<Object, Object> targetDataSources = new HashMap<>(2);
9       targetDataSources.put(DataSourceNames.SHADOW, shadowDataSource);
10      targetDataSources.put(DataSourceNames.MASTER, masterDataSource);
11        // 还有数据源,在targetDataSources中继续添加
12        log.info("DataSources:" + targetDataSources);
13        return new DynamicDataSource(masterDataSource, targetDataSources);
14    }
```

做好以上这些准备工作后，我们就可以开始实现具体的标记识别和数据库隔离动作了。

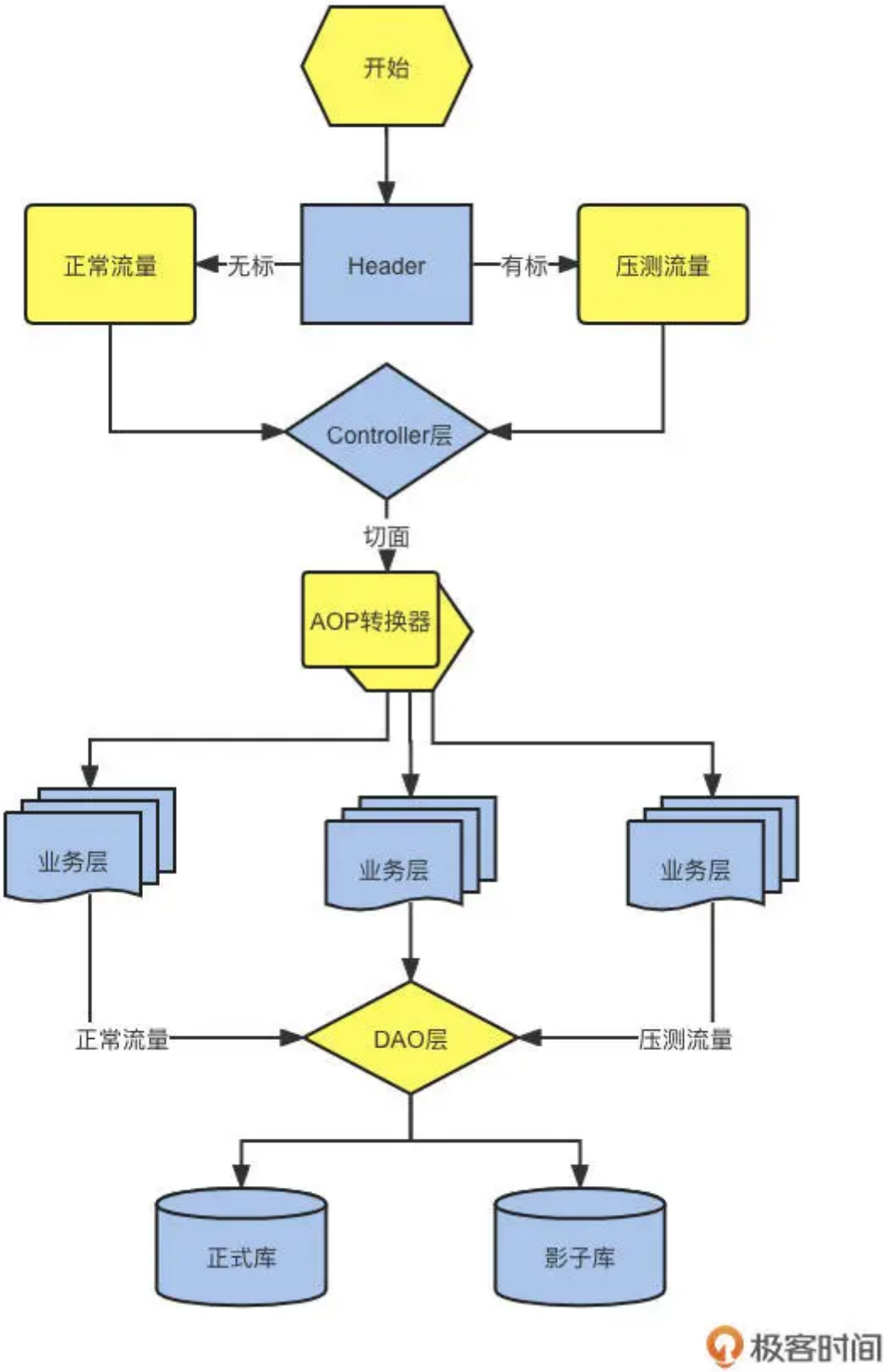
下面，我们主要做以下两种获取标记方案的技术预演：

HttpRequest Header：从 HttpRequest Header 中获取标记，适合单服务、单一 HTTP 协议的场景；

数据上下文：从数据上下文对象中获取标记。**这是更为推荐的微服务标记透传方案。**

第一种：HttpRequest Header 方案

主要逻辑如下图所示：



数据源注入上下文后，刚才提到的 AOP 就派上用场了。这里你需要先学习 AOP 切面编程，使用切面编程拦截并获取 Header 信息。再结合 TransmittableThreadLocal 的特性进行数据源切换。

复制代码

```java
package com.dunshan.data.config;

import lombok.extern.log4j.Log4j2;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

import javax.servlet.http.HttpServletRequest;
import java.util.Arrays;

/**
 * @author dunshan
 * @description: 数据源切换
 * @date 2021-08-15 12:46:32
 */

@Log4j2
@Aspect
@Component
public class DataSourceAspect {



    /**
     * 切点: 所有配置 DataSource 注解的方法
     */
    @Pointcut("execution(public * com.dunshan.data.controller..*.*(..))")
    public void controllerAspect() {
    }

    // 请求method前打印内容
    @Before(value = "controllerAspect()")
    public void methodBefore(JoinPoint joinPoint) {
        ServletRequestAttributes requestAttributes = (ServletRequestAttributes
                .getRequestAttributes();
        HttpServletRequest request = requestAttributes.getRequest();

         //获取 Header 标记
        String header = request.getHeader("dunshan");

        // 打印请求内容
        log.info("===============请求内容===============");
        log.info("请求地址:" + request.getRequestURL().toString());
        log.info("请求方式:" + request.getMethod());
        log.info("请求类方法:" + joinPoint.getSignature());
        log.info("请求类方法参数:" + Arrays.toString(joinPoint.getArgs()));
        log.info("请求类方法 Header 信息:" + header);
```

```
52          // 通过 Header 信息判断
53          if (header != null && header.equals("7DGroup")) {
54                DynamicDataSource.setDataSource(DataSourceNames.SHADOW);
55          } else {
56                DynamicDataSource.setDataSource(DataSourceNames.MASTER);
57          }
58
59      }
60
```

我们可以看到，这里根据 Header 信息中包含的关键字“7DGroup”做流量判断。

上面的技术预演工作已经包含了整个 demo 的关键部分，之后就要验证 demo 能不能运行成功了。如果能成功，就可以移植到目前的工程中去了。

结果验证

好了，我们这就来验证一下。

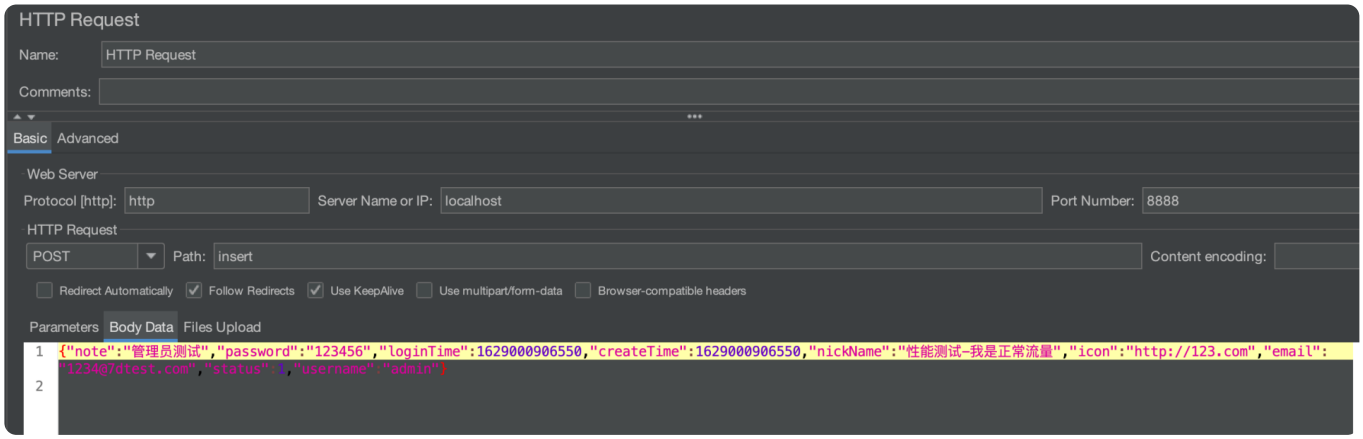
先验证数据源读取是否正常：启动工程，查看日志。

```
2021-08-15 12:22:46.880  INFO 81832 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8888 (http)
2021-08-15 12:22:46.896  INFO 81832 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-08-15 12:22:46.896  INFO 81832 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.30]
2021-08-15 12:22:46.953  INFO 81832 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2021-08-15 12:22:46.953  INFO 81832 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 1200 ms
2021-08-15 12:22:47.182  INFO 81832 --- [           main] c.d.data.config.DynamicDataSourceConfig  : DataSources:{slave={
    CreateTime:"2021-08-15 12:22:47",
    ActiveCount:0,
    PoolingCount:0,
    CreateCount:0,
    DestroyCount:0,
    CloseCount:0,
    ConnectCount:0,
    Connections:[
    ]
}

[
], master={
    CreateTime:"2021-08-15 12:22:47",
    ActiveCount:0,
    PoolingCount:0,
    CreateCount:0,
    DestroyCount:0,
    CloseCount:0,
    ConnectCount:0,
    Connections:[
    ]
}

[
]}
```

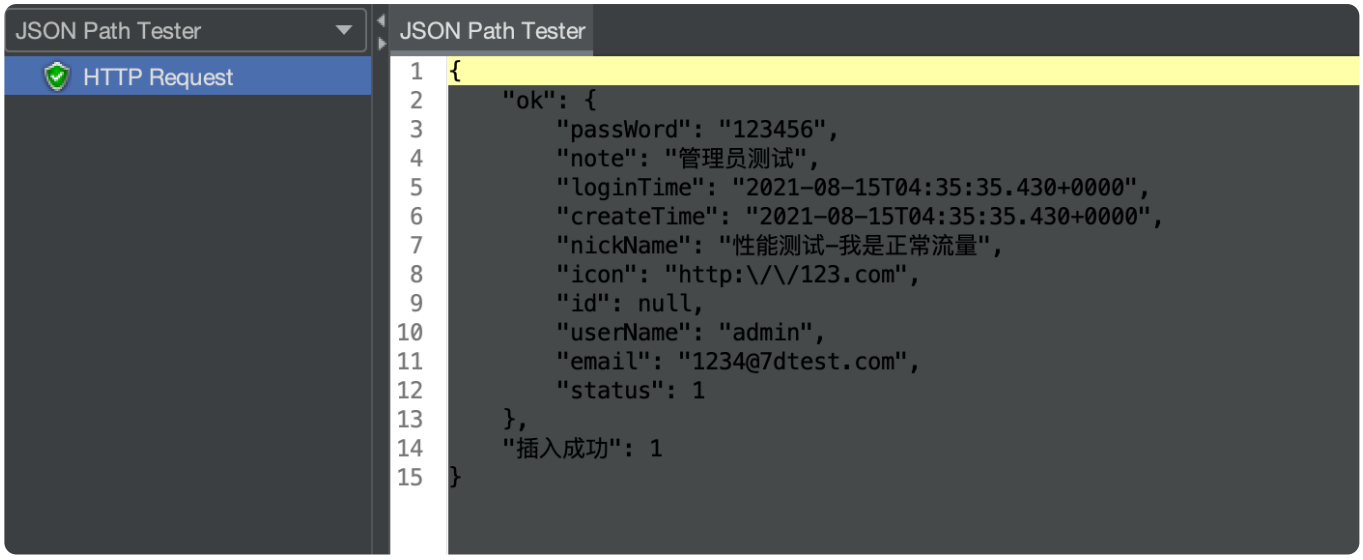
观察启动信息，标示两个数据源已经读取成功。好，接下来就要验证操作能不能成功了。

正式请求（未加标记的请求）

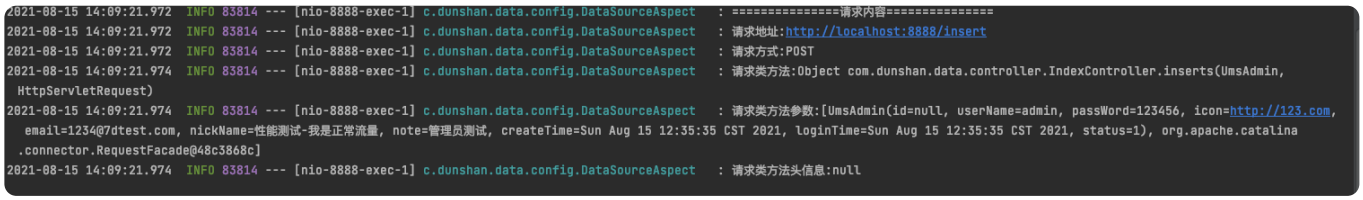
上面的代码已经修改完成了，下面我们要来验证一下是否修改成功。目前我们使用 JMeter 做验证工具，在 JMeter 中增加插入接口请求，并且写入下面数据：



在 body 中的“昵称”中写入“性能测试 - 我是正常流量”，请求成功显示如下：



为了验证是否插入成功，需要打开 MySQL 输入查询语句。在查询前，要先通过日志判断出要使用哪个数据库，这样才能查到结果。

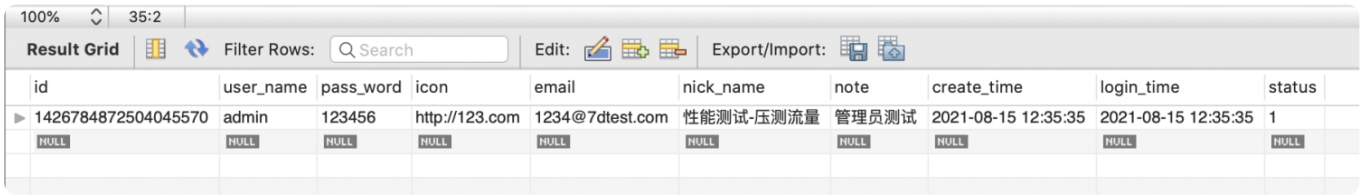


上面的日志中， Header 信息为 null，根据早先的设计， Header 信息为 null 会走 master（正式库） 数据源 ，也就是 mall_master 库。

接下来，打开数据库工具执行 SQL，验证数据是不是已经插入成功。

```
1 SELECT * FROM mall_master.ums_admin;
```

复制代码



100% 35:2

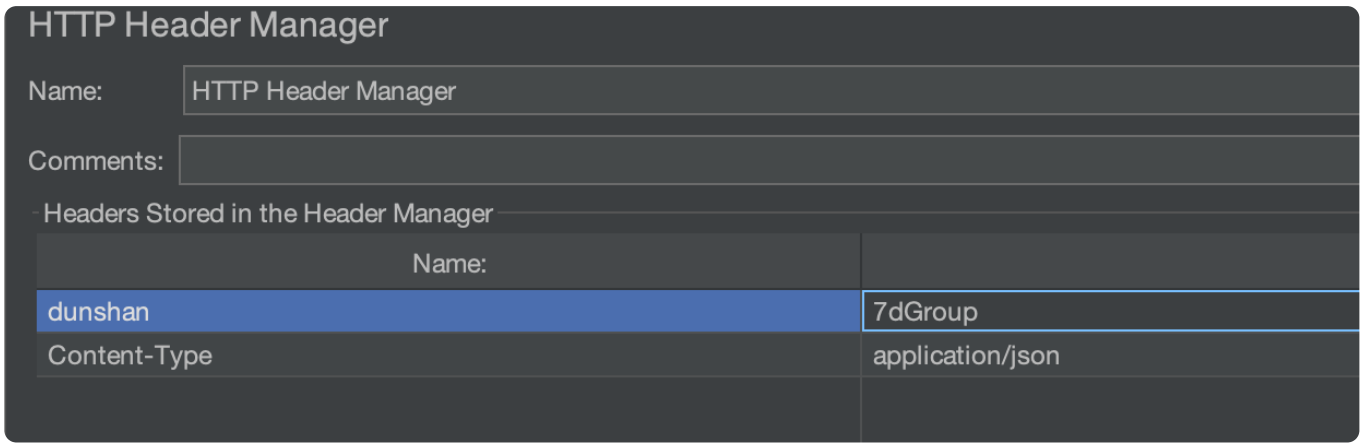
Result Grid | Filter Rows: Search | Edit: | Export/Import:

id	user_name	pass_word	icon	email	nick_name	note	create_time	login_time	status
1426784872504045570	admin	123456	http://123.com	1234@7dtest.com	性能测试-压测流量	管理员测试	2021-08-15 12:35:35	2021-08-15 12:35:35	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

从界面截图中可以看出，显示的数据与压力工具执行的数据一致。

压测请求（加标记的请求）

在 JMeter 中增加 HTTP Header Manager 组件，并且在 Header 信息增加如下标志，再次执行操作，验证数据是否进入压测数据库。如果成功，说明目前的 demo 是有效的。



HTTP Header Manager

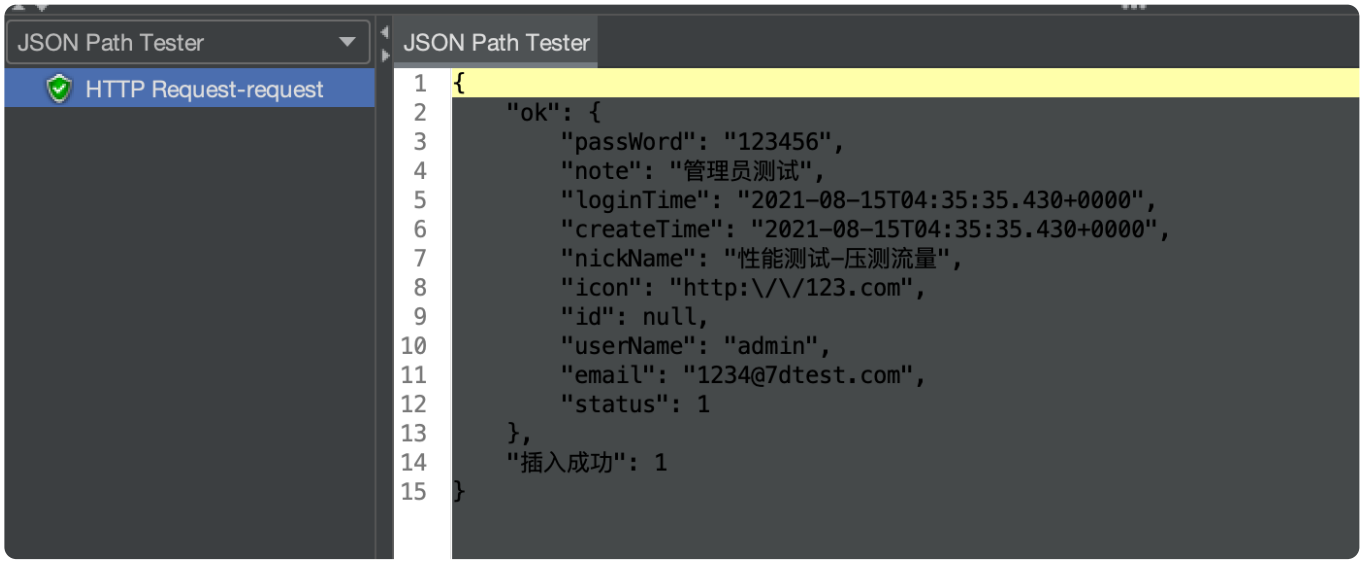
Name: HTTP Header Manager

Comments:

Headers Stored in the Header Manager

Name:	
dunshan	7dGroup
Content-Type	application/json

打开 JMeter，在 View Results Tree 中查看结果。



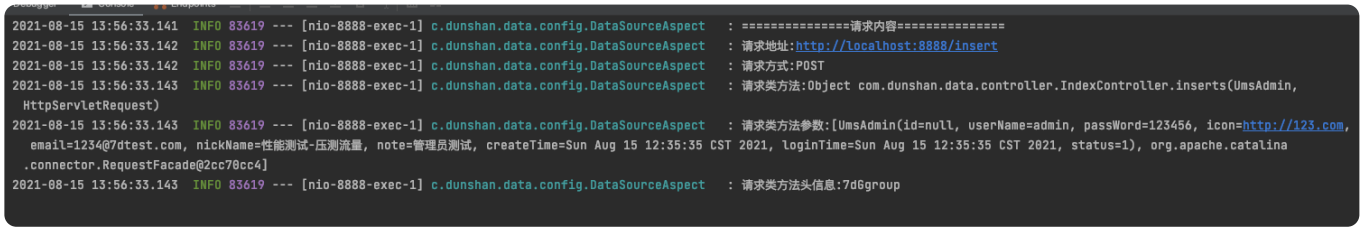
JSON Path Tester

HTTP Request-request

JSON Path Tester

```
{
    "ok": {
        "passWord": "123456",
        "note": "管理员测试",
        "loginTime": "2021-08-15T04:35:35.430+0000",
        "createTime": "2021-08-15T04:35:35.430+0000",
        "nickName": "性能测试-压测流量",
        "icon": "http:\/\/123.com",
        "id": null,
        "userName": "admin",
        "email": "1234@7dtest.com",
        "status": 1
    },
    "插入成功": 1
}
```

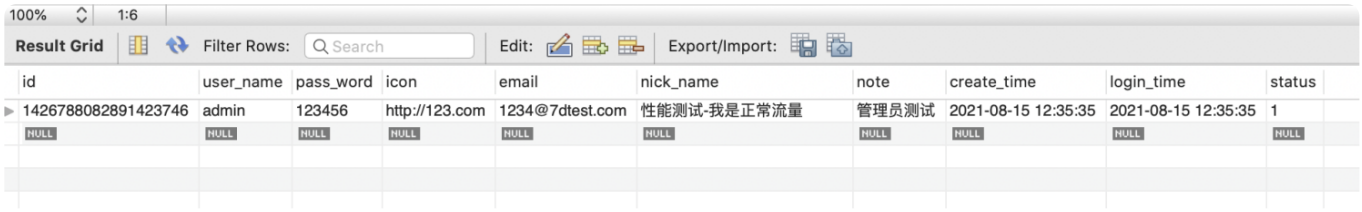
在 SpringBoot 工程中查看请求日志，验证是否成功切换数据源。日志显示目前的标记为 7DGroup ，所以应该走 shadow（影子库）数据源，也就是数据源中的 mall_shadow 库。



根据日志显示的信息，打开 mall_shadow 数据库执行 SQL ，验证数据是否已经插入该数据库。

```
1 SELECT * FROM mall_shadow.ums_admin;
```

复制代码

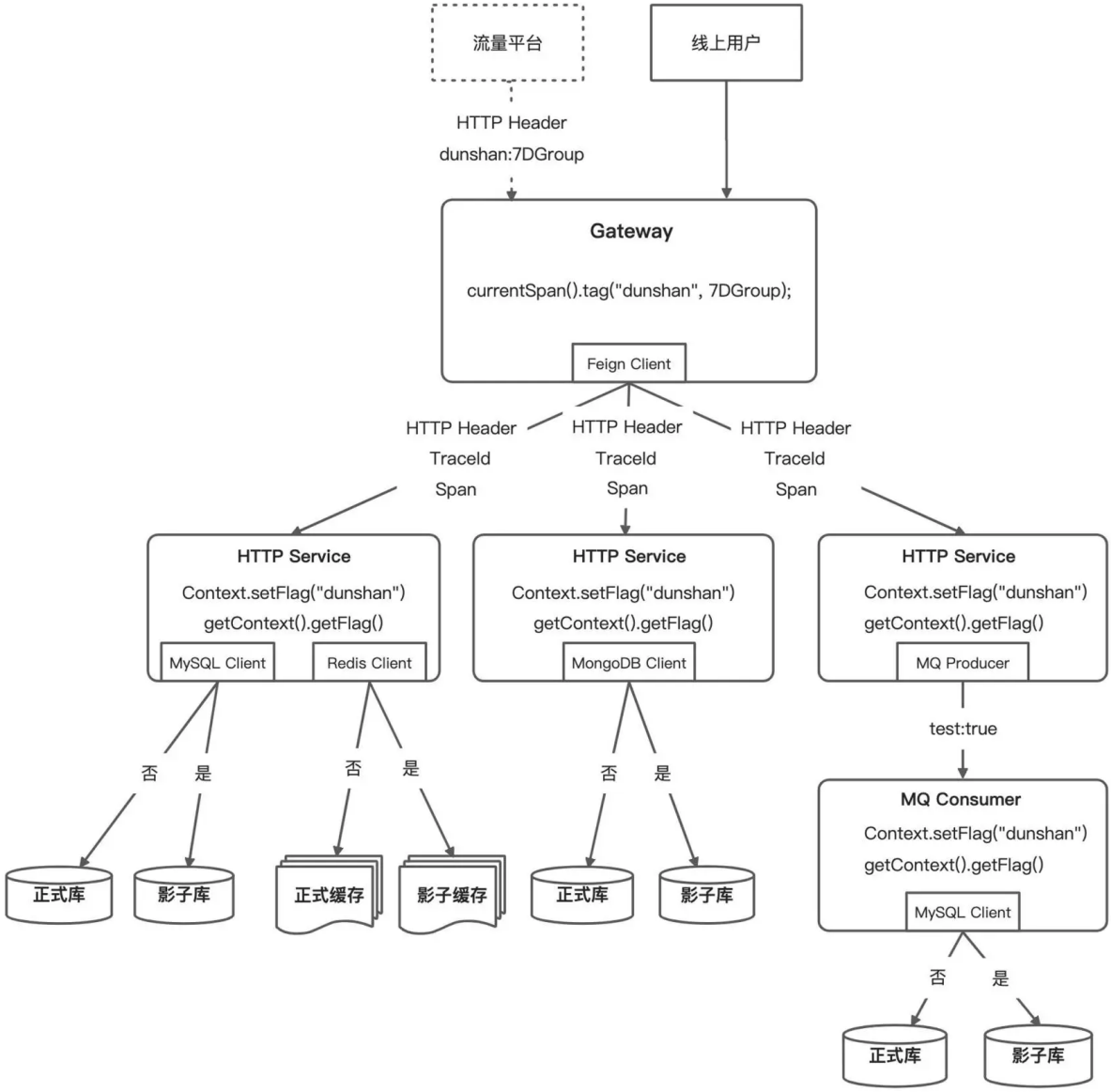


从上面的结果可以看出，目前的数据已经进入预期的数据库，也达成了 demo 技术预演目标。之前已经说过，只要 demo 能完成数据库切换，并且数据正常，那么就需要把目前的配置文件和相关类移植到正式系统中去，只有这样才能低风险完成系统改造。

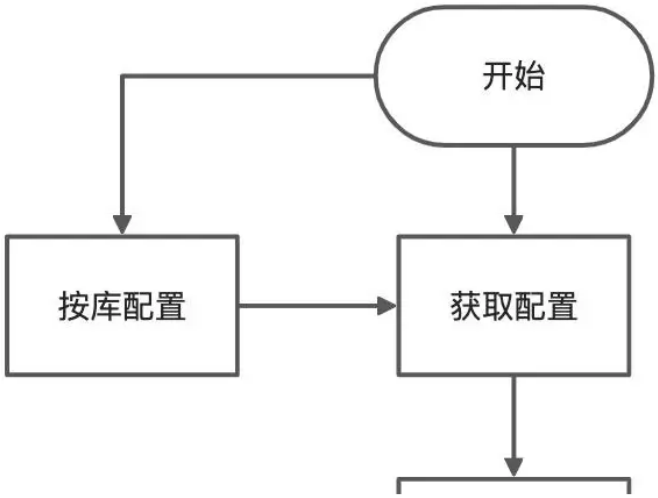
第二种：数据上下文方案

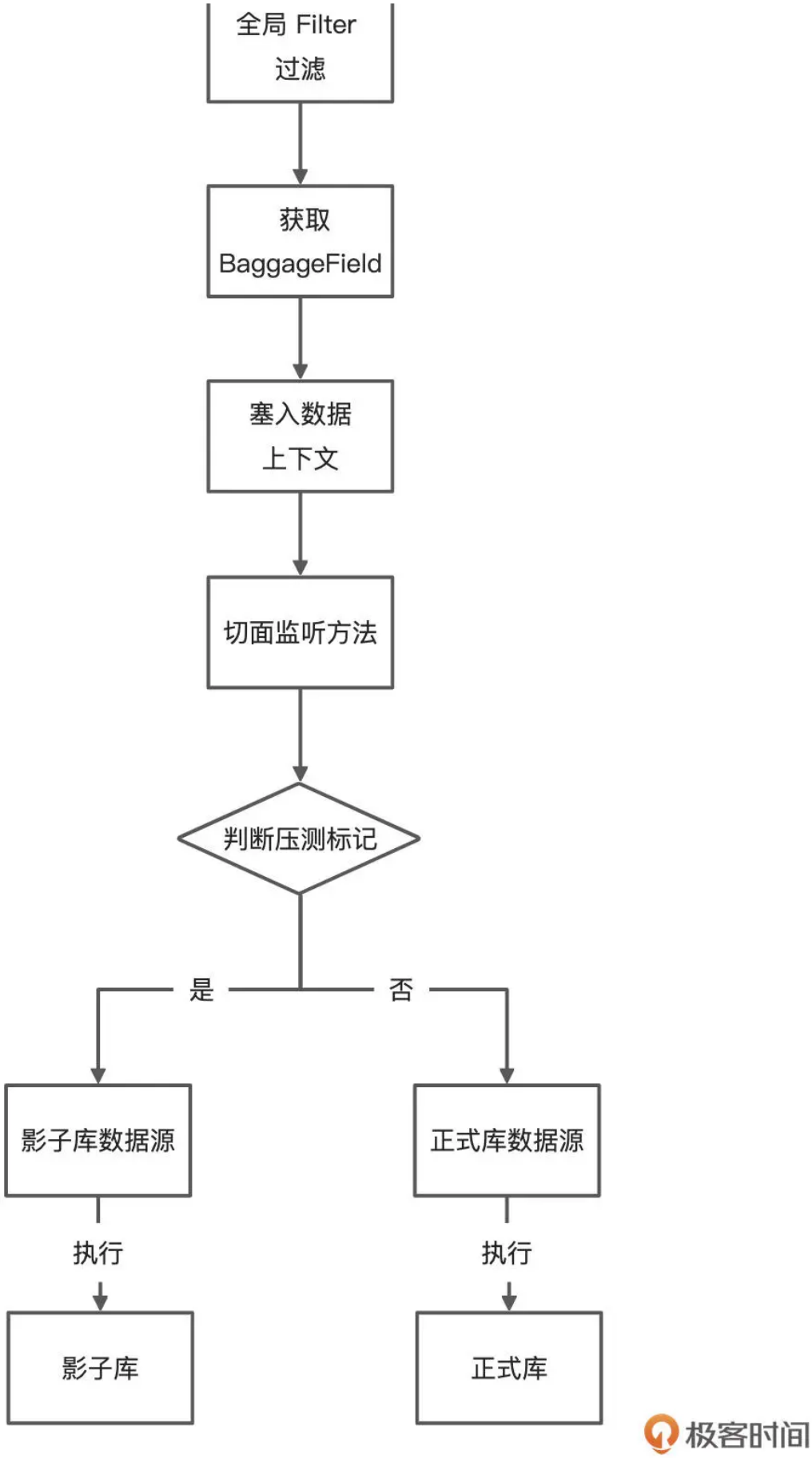
在第一种方案中，最关键的一点是要通过 AOP 拦截切换，而我们下面这个方案，则是通过数据上下文切换获取压测标记和 AOP 切换数据源，其它的内容都是一样的。

在开始演示之前，我们先回顾一下上一讲提到过的方案图：



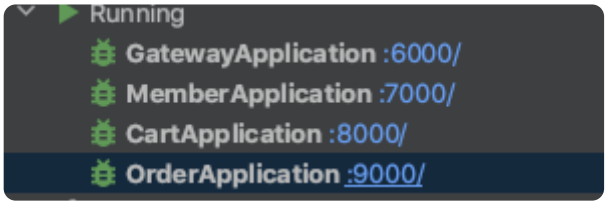
这里实现的核心逻辑图如下：





我们还是使用上一节课搭建的环境，如果忘了这个环境怎么搭，你可以到上一讲复习一下。

首先，启动项目，显示如下：



可以看到，Nacos 注册中心有网关、会员、购物车、订单服务。

服务列表 | public
服务名称 请输入服务名称
分组名称 请输入分组名称
隐藏空服务:
查询
创建服务
服务名 分组名称 集群数目 实例数 健康实例数 触发保护阈值 操作
dunshan-member DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
dunshan-cart DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
dunshan-gateway DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
dunshan-order DEFAULT_GROUP 1 1 1 false 详情 | 示例代码 | 订阅者 | 删除
每页显示: 10
上一页
1
下一页

在上一节课，我们已经把标记传入到了每个服务中，同时还存入了对应的数据上下文里，所以在这里，我们只要在业务层获取标记，判断是不是压测标记，然后做对应的数据源切换动作就可以了。

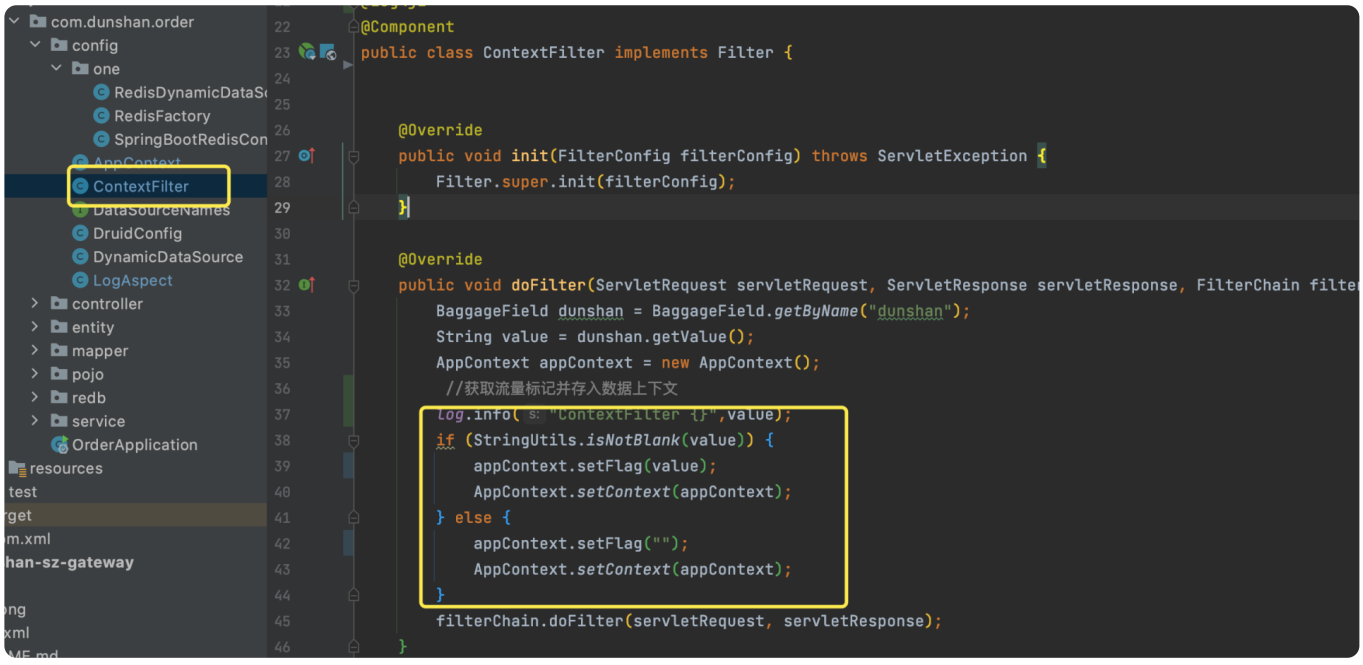
这里，我们打开上一节课的项目，在订单服务中的 application.yml 文件中添加双数据库链接，你可以参考下面的配置：

复制代码

```
spring:
  datasource:
    master: # 数据源1
      username: root
      password: dunshan123root
      url: jdbc:mysql://localhost:3306/mall_master?useUnicode=true&characterEn
      type: com.alibaba.druid.pool.DruidDataSource
      initialSize: 5
      minIdle: 5
      maxActive: 20
      maxWait: 60000
      timeBetweenEvictionRunsMillis: 60000
      minEvictableIdleTimeMillis: 300000
      validationQuery: SELECT 1 FROM DUAL
      testWhileIdle: true
      testOnBorrow: false
      testOnReturn: false
```

```
    poolPreparedStatements: true
    #    配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙
    filters: stat,wall,log4j
    maxPoolPreparedStatementPerConnectionSize: 20
    useGlobalDataSourceStat: true
    connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=
  shadow: # 数据源2
    url: jdbc:mysql://localhost:3306/mall_shadow?useUnicode=true&characterEn
    username: root
    password: dunshan123root
    druid:
      initial-size: 10 #连接池初始化大小
      min-idle: 10 #最小空闲连接数
      max-active: 20 #最大连接数
      web-stat-filter:
        exclusions: "*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*" #不统计这些
      stat-view-servlet: #访问监控网页的登录用户名和密码
        login-username: druid
        login-password: druid
```

想想上节课我们讲的，应该在什么地方获取和设置标记。这样可以方便下一步操作。具体操作可以参考我在文稿中给出的截图。



简单解释下，这里我们添加了一个全局 Filter 拦截器，拦截全部请求，通过 BaggageField.getByName(“dunshan”) 获取透传标记，再把标记放入数据上下文中。

接下来，我们查看一下数据上下文类，这个类的主要目的是把压测标记保存到 TransmittableThreadLocal 中去。

复制代码

```
1
2  /**
3   * @author dunshan
4   * @description: 数据上下文
5   * @date 2021-11-18 23:12:10
6   */
7  public class AppContext implements Serializable {
8      private static final TransmittableThreadLocal<AppContext> contextdunshan =
9      private String flag;
10     public static AppContext getContext() {
11         return contextdunshan.get();
12     }
13     public static void setContext(AppContext context) {
14         contextdunshan.set(context);
15     }
16     public static void removeContext() {
17         contextdunshan.remove();
18     }
19     public String getFlag() {
20         return flag;
21     }
22     public void setFlag(String flag) {
23         this.flag = flag;
24     }
25 }
26
```

下一步还是使用 AOP 判断请求，这次，我们通过数据上下文获取标记来切换数据源链接，你可以参考我给出的代码：

复制代码

```
1  @Log4j2
2  @Component
3  @Aspect
4  public class AopMyDbSwitch {
5      /**
6       * 拦截入口下所有的 public方法
7       */
8      @Pointcut("execution(public * com.dunshan.order.controller..*(..))")
9      public void pointCutAround() {
10     }
11     /**
12      *根据数据上下文切换数据源
13      */
14     @Before(value = "pointCutAround()")
15     public void around(JoinPoint point) {
```

```
        AppContext context = AppContext.getContext();
        String flag = context.getFlag();

        if (StringUtils.isNotEmpty(flag) && flag.equals(DataSourceNames.HEAD))
            //影子库
            MDC.put("dunshan", "shadow");
            DynamicDataSource.setDataSource(DataSourceNames.SHADOW);
        } else {
            //正式库
            MDC.put("dunshan", "produce");
            DynamicDataSource.setDataSource(DataSourceNames.MASTER);
        }
    }
}
```

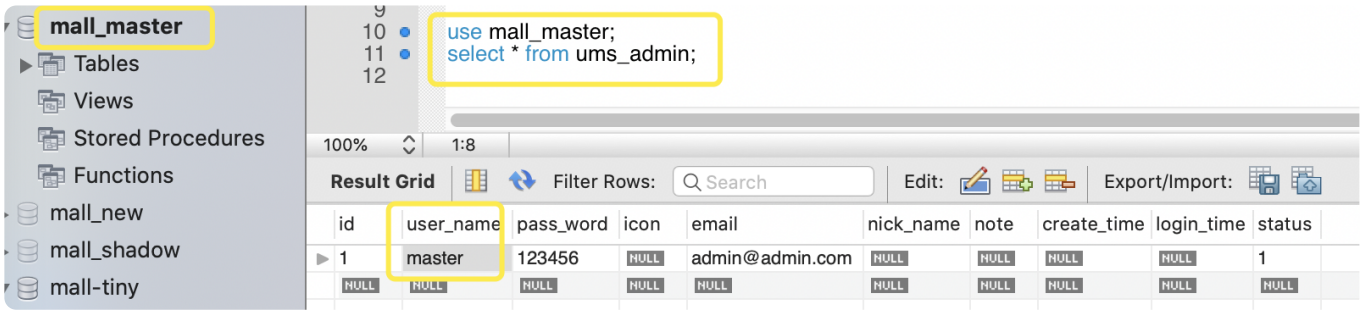
注意：除了刚才所讲的这些不一样的地方以外，其他配置与第一种方案都是一样的。

有上面的配置后，我们再查看一下数据库表是否做了区分。先看 mall_master 中的 ums_admin 表，再看 mall_shadow 中的 ums_admin 表。

mall_master 数据库中 ums_admin 表显示一条内容为 master：

复制代码

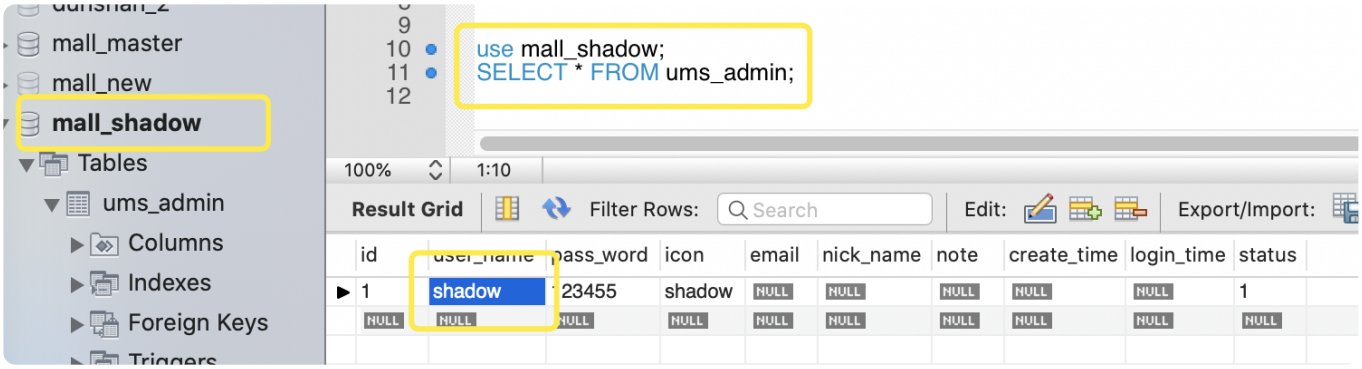
```
use mall_master;
select * from ums_admin;
```



mall_shadow 数据库中的 ums_admin 数据显示有一条记录包含 shadow：

复制代码

```
use mall_shadow;
SELECT * FROM ums_admin;
```



数据库区分完毕后，我们再设计一个请求，用来模拟正式流量与压测流量查询这两个表，在会员系统中增加查询请求，参考如下：

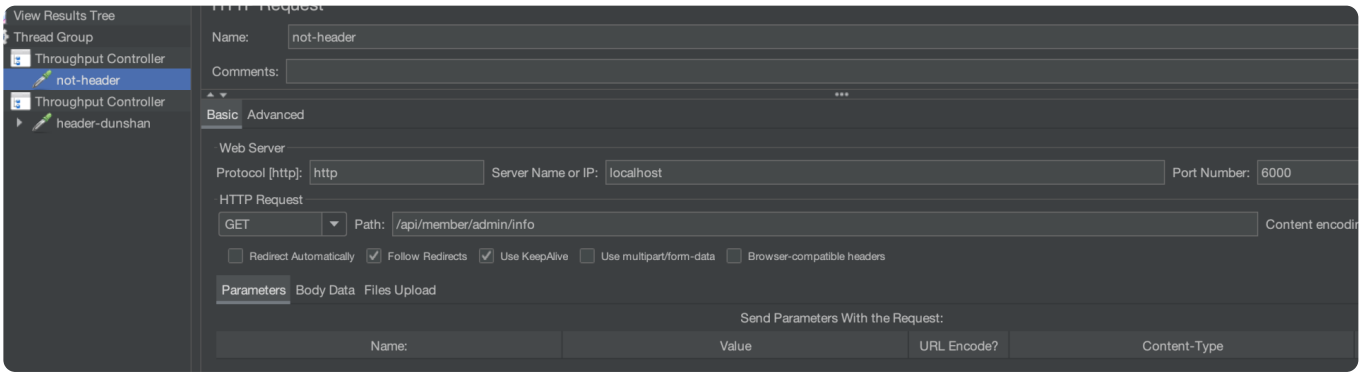
复制代码

```
/**
 * 查询用户信息
 *
 * @return
 */
@GetMapping("/admin/info")
public Object selectAdmin() {
    return cartFeignClient.selectInfo();
}
```

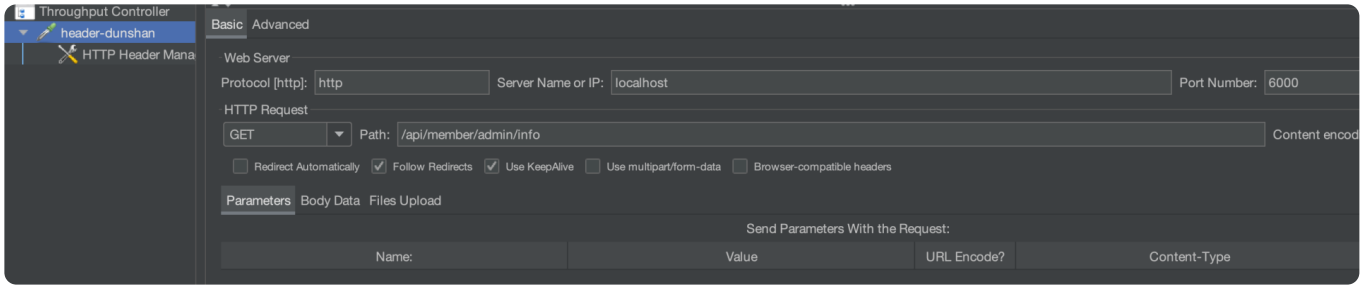
链路调用关系为网关 -> 用户 -> 购物车 -> 订单 -> 数据库。

这里我们还是使用 JMeter 模拟压测流量与正常流量，你可以参考我给出的请求信息。

不添加请求 Header 标记（正式请求）



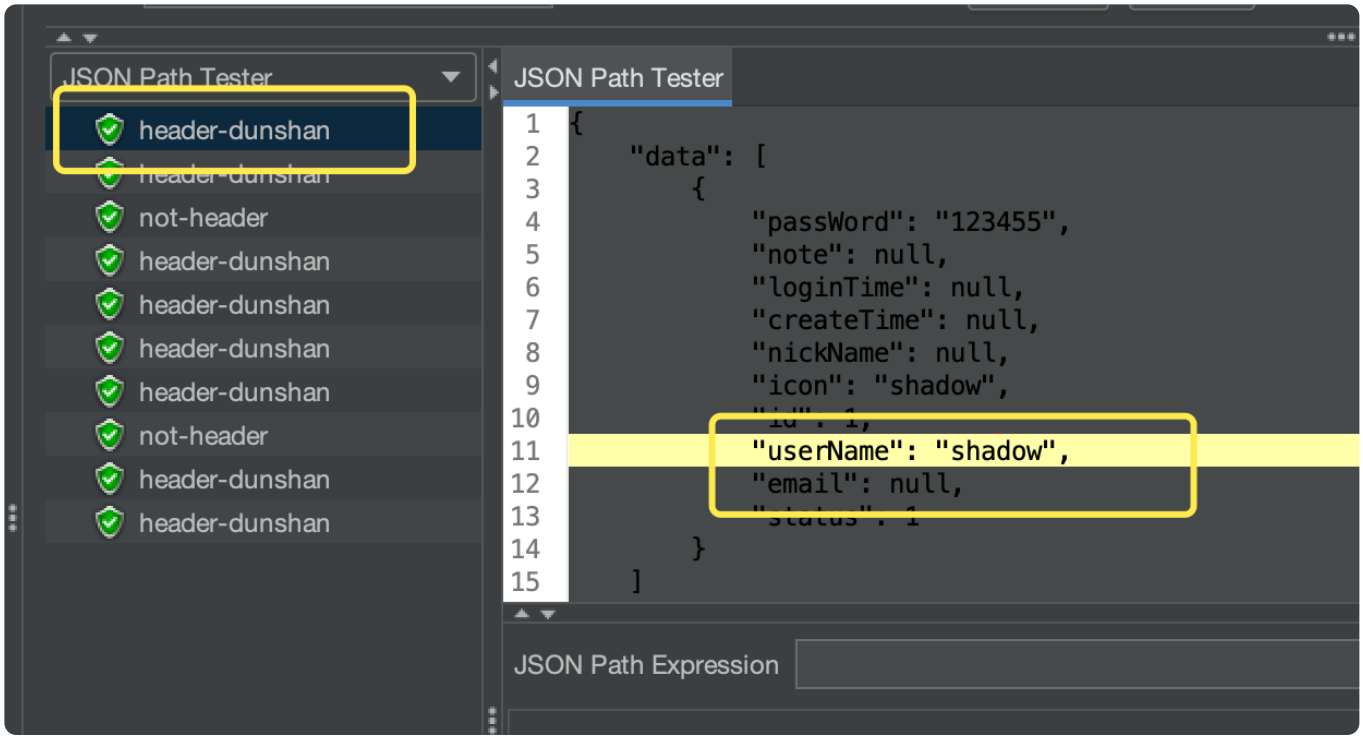
添加请求 Header 标记（压测请求）



接下来，我们要验证结果是否跟预期想的一致。点击 JMeter 请求，在响应结果中查看结果。

不添加请求 Header 标记（正式请求）

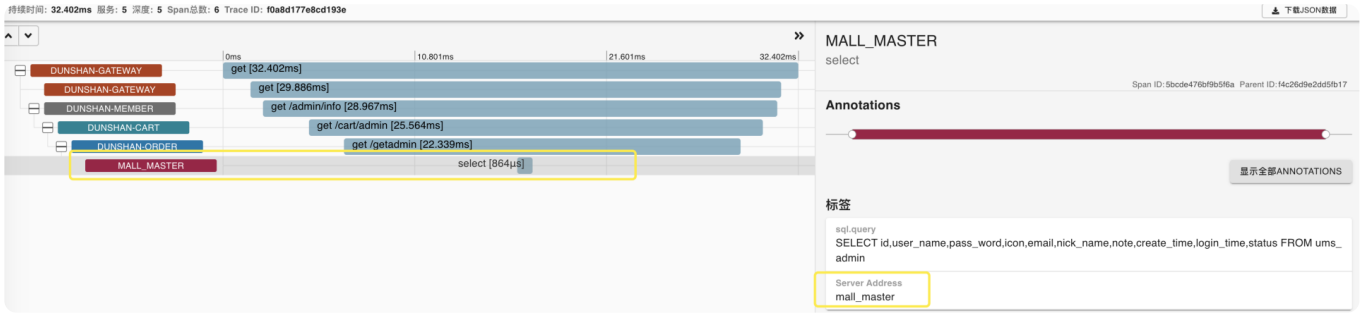
添加请求 Header 标记（压测请求）



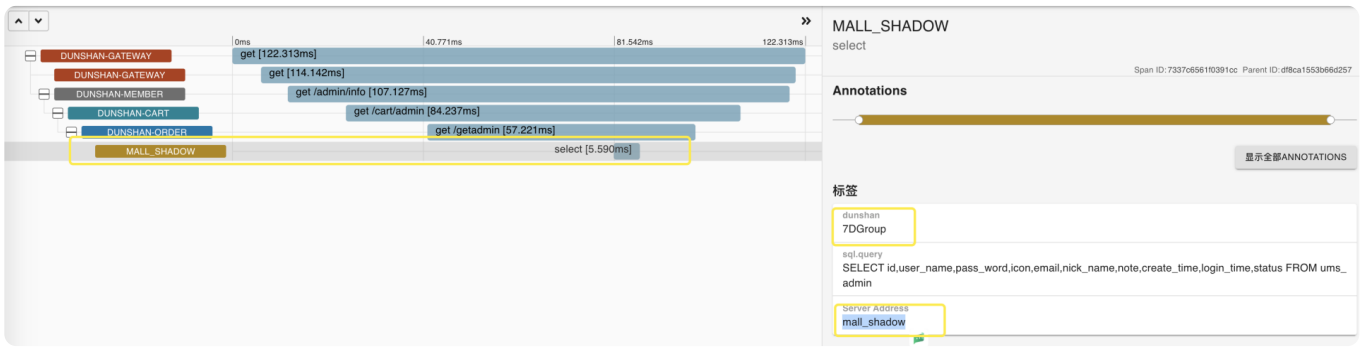
可以看到，数据已经按预期显示出来了。

我们再打开 Zipkin 连接跟踪，验证下链路追踪结果。

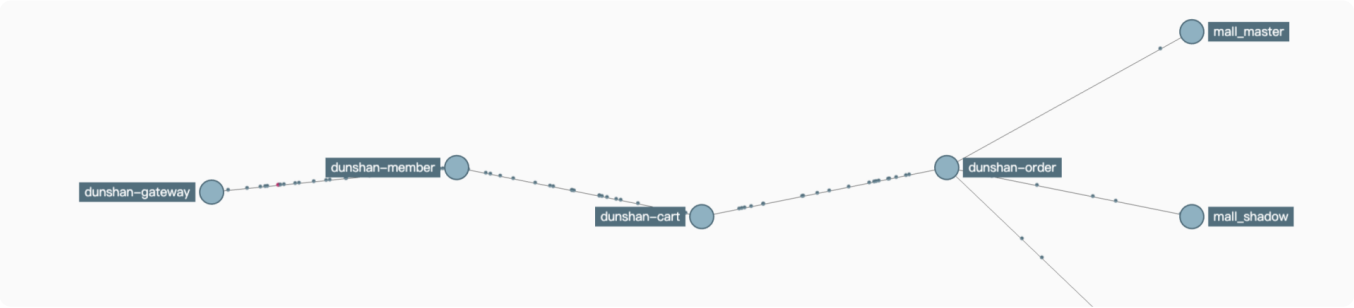
不添加请求 Header 标记（正式库）



添加请求 Header 标记（影子库）



Zipkin 链路全貌图为：



从上面的结果可以看出，目前的数据已经进入到了预期的数据库中，也达成了通过获取数据上下文切换数据库技术预演的目标。之前已经说过了，只要 demo 能完成数据库切换，并且数据正常，那么就可以把目前的配置文件和相关类移植到正式系统中去了。

系统改造

接下来我们就看看真实系统的改造。

首先对 mall-member（会员系统）进行改造。接下来我们快速把 demo 的类与配置移植到对应工程中去。

添加全局 Filter 过滤器

因为我们上一讲已经演示过在全局 Filter 中把标记加入到数据上下文中的逻辑，所以这里的改造就比较轻松了。我们可以快速在服务中添加全局 Filter 过滤器 。

```
@Component
public class ContextFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        Filter.super.init(filterConfig);
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servle
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        String header = request.getHeader(DataSourceNames.DUNSHAN);
        AppContext appContext = new AppContext();
        if (StringUtils.isNotEmpty(header)) {
            appContext.setFlag(header);
            AppContext.setContext(appContext);
        } else {
            AppContext.setContext(appContext);
        }
        filterChain.doFilter(servletRequest, servletResponse);
    }

    @Override
    public void destroy() {
        AppContext.removeContext();
        Filter.super.destroy();
```

配置 AOP 切面

然后，在 AOP 类中修改 Pointcut 类中的注解。需要注意修改中间的文本框，其它地方与 demo 中保持一致即可。

```
/** 切点: 所有配置 DataSource 注解的方法 */
@Pointcut("execution(public * com.dunshan.mall.member.controller..*.*(..))")
public void controllerAspect() {
}
```

具体的代码你可以参考我给出的图片。

```
 */

@Pointcut("execution(public * com.dunshan.mall.member.controller..*.*(..))")
public void controllerAspect() {
}

/**
 * Mysql 数据库切换
 * @param joinPoint join
 */
@Before(value = "controllerAspect()")
public void methodBefore(JoinPoint joinPoint) {

    ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContext
            .getRequestAttributes();
    HttpServletRequest request = requestAttributes.getRequest();
    //获取头标示
    String header = request.getHeader(DataSourceNames.DUNSHAN);
    // 打印请求内容
    log.info( s: "===============请求内容===============");
    log.info( s: "请求地址:" + request.getRequestURL().toString());
    log.info( s: "请求方式:" + request.getMethod());
    log.info( s: "请求类方法:" + joinPoint.getSignature());
    log.info( s: "请求类方法参数:" + Arrays.toString(joinPoint.getArgs()));
    log.info( s: "请求类方法头信息:" + header);


    AppContext context = AppContext.getContext();
    String flag = context.getFlag();
    if (StringUtils.isNotEmpty(flag) && flag.equals(DataSourceNames.HEAD)) {
        MDC.put( key: "dunshan",  val: "shadow");
        log.info( s: "Mysql 压测流量");
        DynamicDataSource.setDataSource(DataSourceNames.SHADOW);
    } else {
        MDC.put( key: "dunshan",  val: "produce");
        log.info( s: "Mysql 真实流量");
        DynamicDataSource.setDataSource(DataSourceNames.MASTER);
    }
```

我们通过 AOP 拦截服务请求，实时判断数据上下文的标记类型，并存入对应的数据源上下文。

配置多数据源

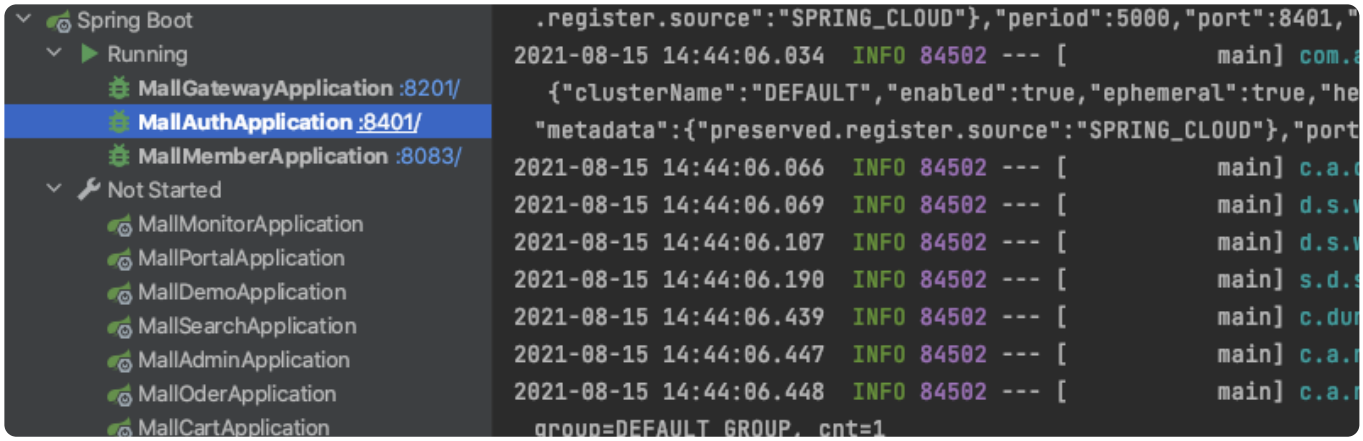
AOP 配置类替换成功后，再把 application.yml 文件修改成多数据源，因为只有这样才能实现数据源切换。具体操作如下：

```yaml
spring:
  datasource:
    master: # 数据源1
      url: jdbc:mysql://60.25.17.9:33333/master?useUnicode=true&characterEncoding=utf-8&serverTimezone=Asia/Shanghai&allowMultiQueries=true
      username: reader
      password: 123456
      druid:
        initial-size: 10 #连接池初始化大小
        min-idle: 10 #最小空闲连接数
        max-active: 20 #最大连接数
        web-stat-filter:
          exclusions: "*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*" #不统计这些请求数据
        stat-view-servlet: #访问监控网页的登录用户名和密码
          login-username: druid
          login-password: druid
    shadow: # 数据源2
      url: jdbc:mysql://39.15.21.27:13306/shadow?useUnicode=true&characterEncoding=utf-8&serverTimezone=Asia/Shanghai&allowMultiQueries=true
      username: reader
      password: 123456
      druid:
        initial-size: 10 #连接池初始化大小
        min-idle: 10 #最小空闲连接数
        max-active: 20 #最大连接数
        web-stat-filter:
          exclusions: "*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*" #不统计这些请求数据
        stat-view-servlet: #访问监控网页的登录用户名和密码
          login-username: druid
          login-password: druid
```

改造完成后，我们启动工程，验证数据库是否成功。因为这个工程比较复杂，所以还需要启动网关服务和认证服务，启动的数据库启动日志如下：

```
2021-08-15 14:38:28.772  INFO 84335 --- [           main] c.d.m.m.c.db.DynamicDataSourceConfig     : DataSources:{slave={
    CreateTime:"2021-08-15 14:38:28",
    ActiveCount:0,
    PoolingCount:0,
    CreateCount:0,
    DestroyCount:0,
    CloseCount:0,
    ConnectCount:0,
    Connections:[
    ]
}, master={
    CreateTime:"2021-08-15 14:38:28",
    ActiveCount:0,
    PoolingCount:0,
    CreateCount:0,
    DestroyCount:0,
    CloseCount:0,
    ConnectCount:0,
    Connections:[
    ]
}}
```

启动系统如下：

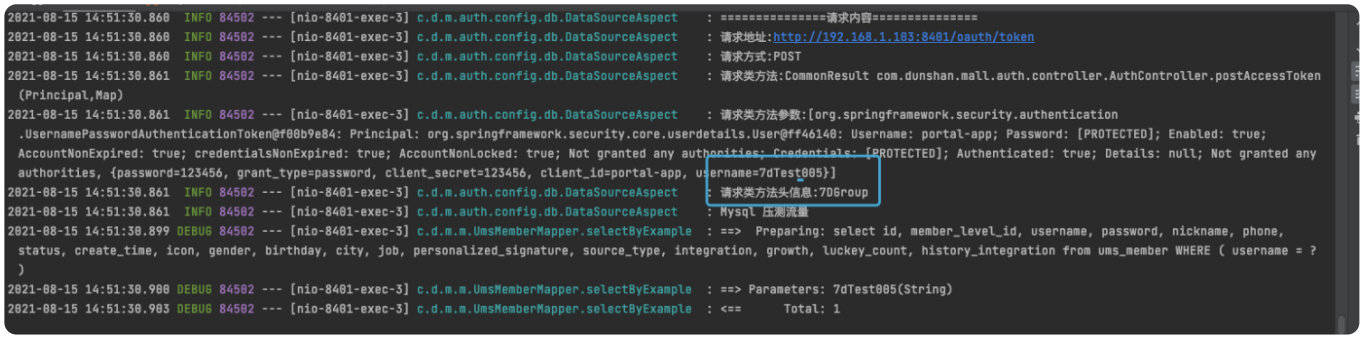


验证改造结果

打开接口文档，在 JMeter 中模拟注册用户接口，关于注册接口怎么开发，你可以参考《高楼的性能工程实战课之脚本开发》，验证 mall-member （会员系统）数据源是否切换成功。

压测请求（带 Header 标记）

脚本编写成功后，执行注册流程，执行成功后打开工程查看日志：



在我给出的日志截图中，7dTest005 是新注册的用户名。7DGroup 是 HTTP Header 标记，这是压测标记，带有它的数据都要进入 shadow（影子）数据库。

根据提示与配置信息，再到数据库中查询数据是否注册成功，仔细观察目前的用户名为 7dTest005。执行 SQL 语句之前，先分析下目前执行的数据库是哪个？

上面的日志信息提示的是压测流量标记，找到数据源为 shadow，执行用户名 SQL 语句。检查一下日志中用户 7dTest005 是否已经注册成功。

复制代码

```
1  select * from shadow.ums_member;
```

100% 1:15

Result Grid | Filter Rows: Search | Edit: | Export/Import:

id	member_level_id	username	password	nickname	phone	status	create_time
1	4	test	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	windir	18061581849	1	2018-08-02 10:35:44
3	4	windy	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	windy	18061581848	1	2018-08-03 16:46:38
4	4	zhengsan	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	zhengsan	18061581847	1	2018-11-12 14:12:04
5	4	lisi	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	lisi	18061581841	1	2018-11-12 14:12:38
6	4	wangwu	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	wangwu	18061581842	1	2018-11-12 14:13:09
7	4	lion	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	lion	18061581845	1	2018-11-12 14:21:39
8	4	shari	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	shari	18061581844	1	2018-11-12 14:22:00
9	4	aewen	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	aewen	18061581843	1	2018-11-12 14:22:55
10	4	7dTest_1_02	$2a$10$/An5Gwd0KRzybaGYYeFjmO/0IU8FkQ...	NULL	13898989879	1	2020-09-25 10:01:37
11	4	7dTest_1_0211	$2a$10$..Q0mCoKDIElLVEsovqQBeNSoGylu7...	NULL	13818989879	1	2020-10-03 16:15:08
12	4	w123456	e10adc3949ba59abbe56e057f20f883e	NULL	18210992071	1	2021-07-23 23:19:55
13	4	7dTest001	e10adc3949ba59abbe56e057f20f883e	NULL	18210992929	1	2021-07-24 11:45:22
14	4	7dTest002	e10adc3949ba59abbe56e057f20f883e	NULL	18210992920	1	2021-07-25 18:25:13
15	4	7dTest003	e10adc3949ba59abbe56e057f20f883e	NULL	18210992921	1	2021-07-25 18:26:06
16	4	7dTest005	e10adc3949ba59abbe56e057f20f883e	NULL	13898989819	1	2021-08-15 14:51:31
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

刚才我们讲的是带 Header 标记的验证，我们根据 SQL 查询结果知道已经注册成功了。这说明流量带压测标记进入了影子库，下面我们再来验证不带标记是否也会正常进入正式库。

正常请求（无 Header 标记）

我们把注册脚本中的 Header 标记去掉，修改用户名执行脚本请求，成功后查看工程日志。如果没有 Header 标记也能成功注册脚本，表示 mall-member 系统改造成功。

执行成功后查看日志：

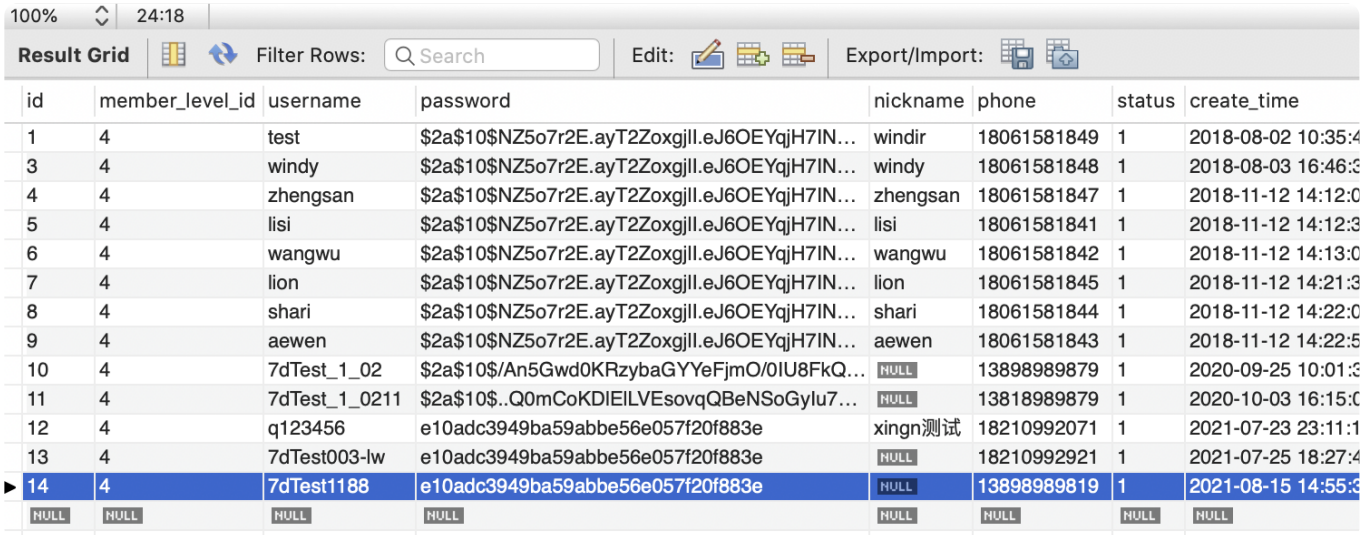
```
2021-08-15 14:55:30.016  INFO 84502 --- [nio-8401-exec-5] c.d.m.auth.config.db.DataSourceAspect    : ===============请求内容===============
2021-08-15 14:55:30.017  INFO 84502 --- [nio-8401-exec-5] c.d.m.auth.config.db.DataSourceAspect    : 请求地址:http://192.168.1.103:8401/oauth/token
2021-08-15 14:55:30.017  INFO 84502 --- [nio-8401-exec-5] c.d.m.auth.config.db.DataSourceAspect    : 请求方式:POST
2021-08-15 14:55:30.017  INFO 84502 --- [nio-8401-exec-5] c.d.m.auth.config.db.DataSourceAspect    : 请求类方法:CommonResult com.dunshan.mall.auth.controller.AuthController.postAccessToken
 (Principal,Map)
2021-08-15 14:55:30.017  INFO 84502 --- [nio-8401-exec-5] c.d.m.auth.config.db.DataSourceAspect    : 请求类方法参数:[org.springframework.security.authentication
 .UsernamePasswordAuthenticationToken@f08b9e84: Principal: org.springframework.security.core.userdetails.User@ff46140: Username: portal-app; Password: [PROTECTED]; Enabled: true;
 AccountNonExpired: true; credentialsNonExpired: true; AccountNonLocked: true; Not granted any authorities; Credentials: [PROTECTED]; Authenticated: true; Details: null; Not granted any
 authorities, {password=123456, grant_type=password, client_secret=123456, client_id=portal-app, username=7dTest1188}]
2021-08-15 14:55:30.017  INFO 84502 --- [nio-8401-exec-5] c.d.m.auth.config.db.DataSourceAspect    : 请求类方法头信息:null
2021-08-15 14:55:30.017  INFO 84502 --- [nio-8401-exec-5] c.d.m.auth.config.db.DataSourceAspect    : Mysql 正常流量
2021-08-15 14:55:30.042 DEBUG 84502 --- [nio-8401-exec-5] c.d.m.m.UmsMemberMapper.selectByExample  : ==>  Preparing: select id, member_level_id, username, password, nickname, phone,
 status, create_time, icon, gender, birthday, city, job, personalized_signature, source_type, integration, growth, luckey_count, history_integration from ums_member WHERE ( username = ?
 )
2021-08-15 14:55:30.043 DEBUG 84502 --- [nio-8401-exec-5] c.d.m.m.UmsMemberMapper.selectByExample  : ==> Parameters: 7dTest1188(String)
2021-08-15 14:55:30.050 DEBUG 84502 --- [nio-8401-exec-5] c.d.m.m.UmsMemberMapper.selectByExample  : <==      Total: 1
```

从我给出的日志截图可以看出，目前流量正常，而且注册的用户名是 7dTest1188，根据代码可知目前走的数据源是 master（正式库）。执行用户 SQL，验证数据是否已经走入 master 数据库。

从下图执行结果可以看出，目前数据已经插入预期的数据库中。

```
1 select * from master.ums_member;
```

复制代码



100% 24:18

Result Grid Filter Rows: Search Edit: Export/Import:

id	member_level_id	username	password	nickname	phone	status	create_time
1	4	test	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	windir	18061581849	1	2018-08-02 10:35:4
3	4	windy	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	windy	18061581848	1	2018-08-03 16:46:3
4	4	zhengsan	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	zhengsan	18061581847	1	2018-11-12 14:12:0
5	4	lisi	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	lisi	18061581841	1	2018-11-12 14:12:3
6	4	wangwu	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	wangwu	18061581842	1	2018-11-12 14:13:0
7	4	lion	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	lion	18061581845	1	2018-11-12 14:21:3
8	4	shari	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	shari	18061581844	1	2018-11-12 14:22:0
9	4	aewen	$2a$10$NZ5o7r2E.ayT2ZoxgjIl.eJ6OEYqjH7IN...	aewen	18061581843	1	2018-11-12 14:22:5
10	4	7dTest_1_02	$2a$10$/An5Gwd0KRzybaGYYeFjmO/0IU8FkQ...	NULL	13898989879	1	2020-09-25 10:01:3
11	4	7dTest_1_0211	$2a$10$..Q0mCoKDIElLVEsovqQBeNSoGylu7...	NULL	13818989879	1	2020-10-03 16:15:0
12	4	q123456	e10adc3949ba59abbe56e057f20f883e	xingn测试	18210992071	1	2021-07-23 23:11:1
13	4	7dTest003-lw	e10adc3949ba59abbe56e057f20f883e	NULL	18210992921	1	2021-07-25 18:27:4
▶ 14	4	7dTest1188	e10adc3949ba59abbe56e057f20f883e	NULL	13898989819	1	2021-08-15 14:55:3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

完成了上述一系列操作后，对 mall-member （会员系统）的改造就完成了。有了成功改造 mall-member 影子库的经验，我们就可以沿用这种方法落地改造到其他系统中去了。

总结

好了，我们今天就讲到这里了。在课程的最后，我再带你回顾一下课程内容。这节课，我们主要讲了落地 MySQL 数据库隔离的方法。

这里有几个要点希望你能记住：

1、多数据源是实现影子库的基础；

2、从数据上下文对象中获取标记，这是更为推荐的微服务标记透传方案；

3、AOP 结合 TransmittableThreadLocal 是实现数据源动态切换的好组合；

4、从风险管控的角度，我建议你先做 demo 技术预演，再做单个系统改造，最后再同步其余系统。

可以说，在全链路压测中，**数据库隔离是最重要的一个改造环节了**。因为做得不到位，压测流量就会污染生产数据，导致的后果非常严重。这节课给出的影子库方案是改造成本低，效果最好的一种方式。希望你能够充分地理解并用好它。

思考题

最后，我想请你思考几个问题：

1. 为什么说数据库隔离是全链路压测中改造的核心环节？
2. 除了我列出的这三种，你还有没有其他方式可以实现数据库隔离？

欢迎你在留言区和我交流讨论。当然，你也可以把这节课分享给你身边的朋友，或许可以碰撞出更多新的想法。我们下节课再见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 14 | 标记透传：如何基于微服务技术进行标记透传？

下一篇 16 | 流量隔离：Redis 缓存隔离是怎么做的？

更多学习推荐

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。